# Examining How the Great Firewall Discovers Hidden Circumvention Servers

Roya Ensafi
Princeton University

David Fifield
UC Berkeley

Philipp Winter
Karlstad & Princeton University

Nick Feamster
Princeton University

Nicholas Weaver
UC Berkeley & ICSI

Vern Paxson
UC Berkeley & ICSI

## ABSTRACT

Recently, the operators of the national censorship infrastructure of China began to employ "active probing" to detect and block the use of privacy tools. This probing works by passively monitoring the network for suspicious traffic, then actively probing the corresponding servers, and blocking any that are determined to run circumvention servers such as Tor.

We draw upon multiple forms of measurements, some spanning years, to illuminate the nature of this probing. We identify the different types of probing, develop fingerprinting techniques to infer the physical structure of the system, localize the sensors that trigger probing—showing that they differ from the "Great Firewall" infrastructure—and assess probing's efficacy in blocking different versions of Tor. We conclude with a discussion of the implications for designing circumvention servers that resist such probing mechanisms.

## Categories and Subject Descriptors

C.2.0 [**General**]: Security and protection (e.g., firewalls); C.2.3 [**Network Operations**]: Network monitoring

## General Terms

Measurement

## Keywords

Active Probing, Deep Packet Inspection, Great Firewall of China, Censorship Circumvention, Tor

## 1. INTRODUCTION

Those in charge of the Chinese censorship apparatus spend considerable effort countering privacy tools. Among their most advanced techniques is what the Tor community terms "active probing": passively monitoring the network for suspicious traffic, actively probing the corresponding servers, and blocking those determined to run circumvention services such as Tor.

*IMC'15,* October 28–30, 2015, Tokyo, Japan.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3848-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2815675.2815690.

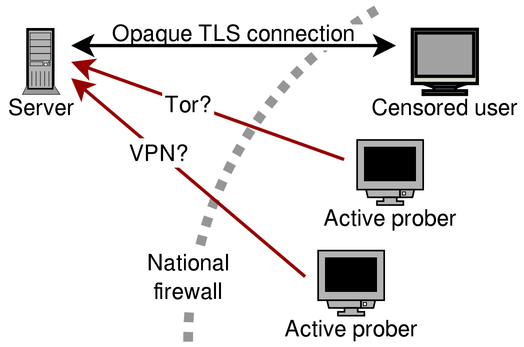


Figure 1: The firewall cannot determine, by mere inspection, whether the encrypted connection carries a prohibited circumvention protocol. Therefore it issues its own probes and observes how the server responds.

The phenomenon of active probing arose presumably in response to enhanced circumvention systems that better resist traditional forms of blocking. For example, instead of employing a protocol recognizable by deep packet inspection (DPI), some of these systems embed their traffic inside TLS streams. Barring any subtle "tells" in the circumvention system's communication, the censor cannot distinguish circumventing TLS from any other TLS, and thus cannot readily block the circumvention without incurring significant collateral damage. Active probing enables the censor to disambiguate the otherwise opaque traffic and once again obtain a measure of control over it.

Figure 1 illustrates the general scheme of active probing. The censor *acts like a user* and issues its own connections to a suspected circumvention server. If the server responds using a prohibited protocol, then the censor takes a blocking action, such as adding its IP address to a blacklist. If the circumvention server does not incorporate access control mechanisms or techniques to distinguish the censor's probes from normal user connections, the censor can reliably identify and block it.

The effectiveness of active probing is reflected in its diverse uses. As of September 2015, researchers have documented its use against Tor [32], SSH [20], and VPN protocols [21, 10], and here we document additional probing targets.

Through this work we aim to better understand the nature of active probing as conducted today against privacy tools and censorship circumvention systems. We seek to answer questions such as: What stimuli cause active probing? How long does it take until a server gets probed? What types of probes do we see, and from where do they originate? How effective is active probing? What does its operation reveal about ways to thwart it?

We consider only "reactive probing:" probing that is triggered by the observation of some stimulus. Censors could also conceivably employ "proactive probing" by scanning the Internet (on a particular port, say) without waiting for a stimulus, but we did not seek to study that. The only possible exception is in our examination of the logs of a server that began to receive active probes without our having instigated them—though the server's status as a Tor bridge may help explain that.

We draw upon a number of datasets from several vantage points, including some extensive longitudinal data, to examine these questions. Our work makes these contributions:

- We describe measurement infrastructure for studying active probing systems.
- We identify various probe types, including some previously undocumented, and chart their volume over time since their first appearance in our data in 2013. The vast majority originate from Chinese IP addresses.
- Using network protocol fingerprinting techniques, we infer the physical structure of the probing system.
- We localize the sensors that trigger active probes and show they are likely distinct from China's main censorship infrastructure, the "Great Firewall" (GFW).

We structure the rest of the paper as follows. Section 2 covers related work, followed by background in Section 3. Section 4 describes our datasets, and Section 5 delves into their analysis; Section 6 concludes.

## 2. RELATED WORK

Academia and civil society have spent significant efforts analyzing and circumventing the GFW, providing us with a comprehensive understanding of how it blocks IP addresses and TCP ports [7], DNS requests [1, 24], and HTTP requests [22, 3]; and the nature of its TCP processing [13].

McLachlan and Hopper [19] warned of the possibility of Tor bridge discovery by Internet scanning in 2009. The study of practical, in-the-wild "active probing" associated with Chinese censorship began in late 2011, when Nixon noticed suspicious entries in his SSH log files [20], including non-conformant payloads of seemingly random byte strings. Careful analysis revealed a pattern: these strange probes, which originated from IP addresses in China, were triggered by prior genuine SSH logins, by real users, from different Chinese IP addresses. In 2012, Wilde documented a similar phenomenon, this time targeting the Tor protocol [30]. Motivated by reports that China was blocking Tor bridges only minutes after their first use from within China, he investigated and observed the GFW performing active probing, triggered by observing a particular list of TLS cipher suites, the one offered by Tor clients. The probing took the form of TLS connections that attempted to establish Tor circuits. Wilde also observed "garbage" random binary probes like the ones seen by Nixon for SSH.

Later in 2012, Winter and Lindskog revisited Wilde's analysis using a server in Beijing [32]. They attracted probers over a period of 17 days and analyzed the probers' IP address distribution, how blocking was effected, and how long bridges remained blocked. They conjectured, but did not establish, that the GFW uses IP address hijacking to obtain its large pool of source IP addresses; that is, that the probing apparatus temporarily borrowed IP addresses that were otherwise allocated.

In 2013, reports suggested that the GFW had begun active probing against obfs2 [31], an obfuscation transport for Tor specifically designed to be difficult to detect by DPI. (A description of obfs2 appears in the next section.) The timing of these reports corresponds well with our own data.

A year later, Nobori and Shinjo discussed their experience with running a large VPN cluster for circumvention [21]. They likewise observed a pattern of connections from China shortly prior to blocking of a server. Other reports indicate that VPN services receive similar probing [10].

Our work aims to broaden the above perspectives, which have generally relied upon one-time measurements from single vantage points; and to illuminate the nature of active probing in greater depth, including its range of probing, response times, and system infrastructure.

## 3. BACKGROUND: CIRCUMVENTION PROTOCOLS

Active probing is a reaction against the increasing effectiveness of censorship circumvention. In this section we briefly describe Tor's place in the world of circumvention, and the obfuscated protocols ("transports") that cloak Tor traffic and make it more resistant to censorship. Three of these protocols—"vanilla" Tor, obfs2, and obfs3—underlie and motivate our experiments. More than that, though, these protocols tell a small part of the story of the global censorship arms race. In their technological advancement, one can see the correspondingly increasing sophistication of censors: starting from their ignorance of Tor, moving on to simple IP address-based blocking, then online detection of obfuscation, and now active probing.

### 3.1 Tor

Years ago, Tor found success in evading various types of censorship such as web site blocks. Censored users found they could treat the network as a simple proxy service with many access points (its anonymity properties being of secondary importance to these users). Despite this success, however, the unadorned "vanilla" Tor protocol is not particularly suited to circumvention. Once censors began looking for it, they found it easy to block. Tor's biggest weakness in this respect is its global public list of relays. A censor can simply download this list and add each IP address to a blacklist—and censors began to do exactly that.

In response to the blocking of its relays, the operators of the Tor network began to reserve a portion of new relays as secret, non-public "bridges." Unlike ordinary relays, bridges are not easily enumerable [6]. They are carefully distributed through rate-limited out-of-band channels such as email and

HTTPS, and only a few at a time. The goal is to make it possible for anyone to learn a few bridge addresses, while making it hard for anyone to learn them all. By design, learning many bridge addresses requires an attacker to control resources such as an abundance of IP addresses and email addresses, or the ability to solve CAPTCHAs.

Even using secret bridge relays, Tor remains vulnerable to detection by deep packet inspection (DPI). Tor uses TLS in a fairly distinctive way that causes it to stand out from other TLS-based protocols. Censors can inspect traffic looking for the "tells" that distinguish Tor from other forms of TLS, and block connections as they arise. After early efforts to make their use of TLS less conspicuous [28], the developers of Tor settled on a more sustainable strategy: wrapping the entire Tor TLS stream in another layer—a "pluggable transport" [29]—that assumes responsibility for protocol-level obfuscation. This model allows for independent innovation in circumvention, while leaving the core of Tor free to focus on its main purpose of anonymity.

## 3.2 Obfs2

The first pluggable transport was *obfs2* [25], introduced in 2012. It was designed as a simple, expedient workaround for DPI of the kind that was then occurring in Iran [4]. It provides a lightweight obfuscation layer around Tor's TLS, re-encrypting the entire stream with a separate key in a way that leaves no plaintext or framing information that can serve as a basis for blocking—the entire communication looks like a uniformly random byte stream in both directions. The simple scheme of obfs2 had immediate success. The protocol has a serious deficiency, though: it is possible to detect it completely passively and with high confidence. Essentially, obfs2 works by first sending a key, then sending ciphertext encrypted with that key. Therefore a censor can simply read the first few bytes of every TCP connection, treat them as a key, and speculatively decrypt the first few bytes that follow. If the decryption is meaningful (matching a TLS handshake, for example), then obfs2 is detected and the censor can terminate the connection.

We turned the weakness of obfs2 to our advantage. Its easy passive detectability, coupled with its lack of use for anything but circumvention (and active probing), meant that we could mine past network logs looking for obfs2 connection attempts. Later we will describe how we used obfs2 probes to seed a list of past prober IP addresses.

## 3.3 Obfs3

The follow-up protocol *obfs3* [26] was designed to remedy this critical flaw in obfs2. Its key innovation is a Diffie-Hellman negotiation that determines the keys to be used to encrypt the rest of the stream. (The key exchange is not as trivial as it may seem, because it, like the rest of the protocol, must be indistinguishable from randomness.) This enhancement in obfs3 deprives the censor of the simple, passive, reliable distinguisher it had for obfs2. The censor must either intercede in the key exchange (using a man-in-the-middle attack to learn the secret encryption keys), or settle for heuristic detection of random-looking streams. While either of these options may be problematic to implement, heuristic detection becomes entirely workable when combined with active probing. An initial, inaccurate test can identify potential obfs3 connections; then an active probe confirms or denies the suspicion.

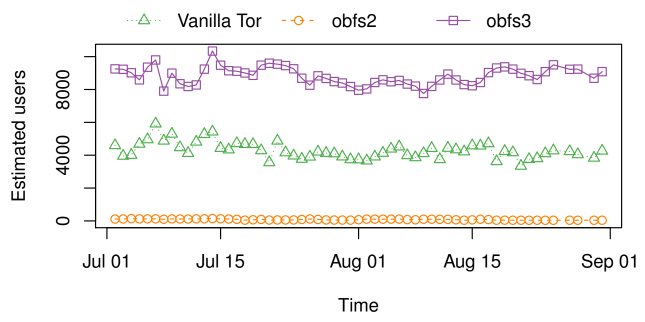


Figure 2: The estimated user numbers of the three transport protocols we study—vanilla Tor, obfs2, and obfs3—in July and August 2015. Obfs3 is the most popular protocol, followed by vanilla Tor. Obfs2 is superseded and sees practically no use any more.

## 3.4 Other Protocols

Though we limited the focus of our active experiments to Tor-related protocols, in the course of gathering data we incidentally found evidence of probing for other protocols, unrelated to Tor except that they also have to do with circumvention. The first of these probes, which we have labeled *AppSpot* in this paper, is an HTTPS-based check for domain fronting [8], a circumvention technique that disguises access to a proxy by making it appear to be access to an innocuous web page. In all of the examples we found, the probes checked whether a server is capable of fronting for Google App Engine at its domain `appspot.com`. The other probe we discovered we label *SoftEther*, because it resembles the client portion of the handshake of SoftEther VPN, the VPN software underlying the VPN Gate circumvention system [21]. Because we found these ancillary types of probe activity by accident, we make no claims to thoroughness in our coverage of them, and suggest that there may be other, yet unknown types of active probing to discover.

Our study focuses on vanilla Tor, obfs2, and obfs3, these being the commonly used protocols that remain vulnerable to active probing. There are other, newer protocols, including spiritual successors ScrambleSuit [33] and obfs4 [27], that have resistance to active probing as an explicit design criterion. Although they are gaining in popularity, they have not yet eclipsed obfs3. The key enhancement of these successor protocols is that they require the client, in its initial message, to prove knowledge of a server-specific secret (transmitted out of band). Put another way, mere knowledge of an IP address and port is not enough to confirm the existence of a circumvention server. As of this writing, obfs3 remains Tor's most-used transport, having around 8,000 simultaneous users on average, as shown in Figure 2. The obfs2 protocol is deprecated, no longer offered in the user interface, and its use is on the wane.

## 4. EXPERIMENTS

We base our results on several experiments, each resulting in a dataset that offers a distinct view into the behavior of active probing. The datasets cover different time ranges (see Table 1) and involve different setups. Table 2 summarizes the phenomena that each can illuminate, with each contributing at least one facet not covered by the others. We now describe each experiment in detail.

| Dataset | Time span |
|---|---|
| Shadow | Dec 2014 – Feb 2015 (three months) |
| Sybil | Jan 29, 2015 – Jan 30, 2015 (20 hours) |
| Log | Jan 2010 – Aug 2015 (five years) |
| Counterprobe | Apr 22 – Apr 27 (six days) |

Table 1: Timeline of our experiments. We created four datasets that span hours, days, months, and years.

|  | Shadow | Sybil | Log | Counter-probe |
|---|---|---|---|---|
| Probing rate |  | ✓ |  |  |
| ISN patterns |  | ✓ |  |  |
| TSval patterns | ✓ | ✓ | ✓ |  |
| obfs2/3 blocking | ✓ |  | ✓ |  |
| Tor bootstrapping | ✓ |  |  |  |
| Probing types |  |  | ✓ |  |
| Architecture |  |  | ✓ | ✓ |
| Topology |  |  |  | ✓ |

Table 2: Observed phenomena and their visibility in our datasets.

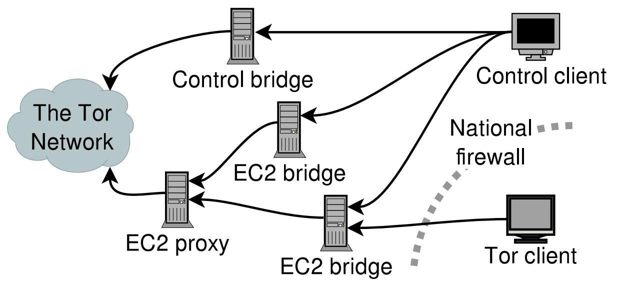


Figure 3: Experimental setup for the "Shadow" dataset.

## 4.1 Shadow Infrastructure

We built a "shadow infrastructure" of our own Tor clients and bridges for a controlled experiment of active probing over time. These clients and bridges were not actually used by any real users, but rather were dedicated exclusively to our own experimental purposes. The infrastructure tested vanilla Tor, obfs2, and obfs3 in equal measure, since active probing is known to target all three of these protocols. Figure 3 illustrates this setup.

We had six Tor clients within China: three in China Unicom, a large country-wide ISP; and three in CERNET, the Chinese Research and Education Network. (We chose CERNET because previous work suggested that censorship of CERNET might differ from the rest of China [7].) Outside of China, we ran six Tor bridges in Amazon's EC2 cloud. Two of the bridges ran vanilla Tor, two ran obfs2, and two ran obfs3. We assigned each of our six clients in China to a unique EC2 machine; the clients never contacted any bridge other than their own assigned one. Initially, all clients attempted to connect to their assigned bridge every 15 minutes. After one month, we changed this to five minutes after preliminary analysis showed that finer granularity in timing might be useful.

We also created a control group consisting of nine bridges (six in Amazon EC2, three in a US university) and a single client outside of China. We never connected to any of the control bridges from one of the clients within China; by comparing the traffic received by our "active" and control bridges, we can isolate general background scanning from active probing by the GFW. The three control bridges not hosted on EC2 allow us to determine whether the GFW treats EC2-hosted servers differently from others. The control client outside of China connects to all of the bridges. If our control client could not establish a Tor connection to one of the "active" bridges, we discarded the measurement we did from China for that bridge.

We took various steps to prevent our bridges from being discovered by any means other than active probing. We configured all of them to be *private bridges*, which means that they did not advertise themselves publicly, neither to the public Tor directory, nor to its database of secret bridges. As a result, no genuine Tor user should attempt to connect to one of our bridges. The bridges listened on random ports in the ephemeral range, to reduce the chance of their discovery by blind Internet scanning. Finally, we used another EC2 machine to proxy the communication between our Tor bridges and the first public Tor relay in a circuit. This extra proxy hop is to prevent another potential bridge-discovery attack, wherein a malicious Tor relay makes a list of all the IP addresses that connect to it (cf. [14, §III.D]).

## 4.2 Sybil Infrastructure

To obtain broader insight into the extent of the censor's active probing infrastructure, we designed another experiment to attract[1] many active probers in a short period of time.

We did so by constructing a "Sybil infrastructure," so named because it seemingly consisted of hundreds of distinct Tor servers. We used a virtual private server (VPS) in France and one in China. We ran a vanilla Tor bridge on the VPS in France and redirected the port range 30000–30600 to our Tor port using firewall port redirection. The actual Tor server ran on a separate port in the ephemeral range.

Then, from our VPS in China, we established Tor connections to every port in the port range in ascending order. This took approximately two hours, because we waited several seconds in between connection attempts. Since the GFW blocks by IP:port tuple, not just IP address [32], the GFW interpreted every single port in the range as a distinct Tor bridge and probed them separately. This experiment resulted in 622 active probing connections (and significantly more TCP connections, as we will discuss later) to the VPS in France.

## 4.3 Server Log Analysis

This dataset comes from the application logs of a server operated by one of the authors, some stretching back to January 2010. The server runs various common network services, including the three we use in the analysis: HTTP, HTTPS, and SSH. In addition to common networking ports, the server has hosted a Tor bridge since January 2011, an ordinary vanilla bridge without pluggable transports. By mining the application logs, we found that the server has been receiving active probes from China for over 2.5 years. An important difference in the Log experiment compared

[1]Our earlier analysis confirmed that simply establishing an initial TLS handshake with a server suffices to attract a prober.

to the others is that it is not the result of artificial activity to induce probing. Our server received its first probes in January 2013, slightly earlier than the first public reports of obfs2 probing [31]. On average, it has received dozens of probes every day since active probing began—though there are long stretches during which it received few probes.

This dataset provides an invaluable longitudinal perspective, though with the significant limitation that application logs do not record as much forensic information as we would like: they omit source ports and other transport-layer information, and usually truncate probe payloads (see below).

*Data Types and Ranges.*
The HTTP and HTTPS log go back to January 2010, before the earliest reports of active probing of any kind. The SSH log dates only to September 2014; probing of that port was already in effect at the beginning of the log. While application logs do not contain as much information as would, say, packet captures, they suffice in many cases to identify the type of probe and the prober's IP address. The HTTPS log is even superior to a packet capture in one respect: for TLS probes it contains the decrypted application data, which would not be accessible from a packet capture.

Along with the application logs, the system has full packet captures for ports 23, 80, and 443, between December 2014 and May 2015. The packet captures enable us to perform more detailed analysis on the network- and transport-layer features of probes. We opened port 23 (Telnet) specially during this period, and used it to host a multi-protocol honeypot server capable of responding to probes of various types (TLS, obfs2, and obfs3), and recording the protocol layers inside them. (To avoid inadvertently capturing potentially sensitive activity, we do not perform any packet capture or additional logging on the SSH and Tor ports.)

The server from which we gathered the Log data is a working server that performs a number of functions apart from simply absorbing active probes. In order to distinguish active probes from operational traffic, we used a form of conservative snowball sampling. We started by extracting incontrovertible probes; namely, obfs2 probes and non-protocol-conforming payloads that we could not otherwise explain, for example random binary garbage written to the HTTP port. (It is easy to detect obfs2, with negligible false positives, because of the protocol weakness described in Section 3.2. It requires only the first 20 bytes sent by the client.) We then made a list of the IP addresses that had sent those probes, and examined all other traffic they had sent at any point in time. Despite that the GFW's active probing rarely reuses source IP addresses, we occasionally found a new probe type. When we did, we added it to our list of known probes and repeated the process. In this way, we slowly expanded our universe of known active prober IP addresses, all the while checking manually to make sure we were not sweeping up non-probing traffic.

This conservative approach enabled us to find a variety of probing behaviors with few false positives, at the potential cost of missing some novel probes from IP addresses that did not also send a recognized probe type. This technique led to our discovery of the "AppSpot" and "SoftEther" probes, even though we seeded the process only with Tor-related probes. Except for a handful of manually excluded hosts (e.g., systems under our control), we did not consider the source IP address in deciding whether a log entry indicated a probe. Specifically, we did not employ IP geolocation to find probes emanating from China, though Figure 7 shows that the probes we found did in fact overwhelmingly come from China.

We found a small amount of non-probing traffic (e.g., ordinary HTTP requests for actual pages) from IP addresses that also sent a probes at some other time. The shortest separation between probe and non-probe was three weeks, and the longest was two years.

*Limitations.*
Our application logs have several limitations. The HTTP and HTTPS (Apache) log truncates at the first '\0', '\n', or '\r\n' sequence, and omits a leading '\n' (we account for this possibility when classifying probes). The SSH (OpenSSH) log truncates at the first '\0', '\r', or '\n', and has a hard limit of 100 bytes.

A significant effect of these limitations is that application logs will not record any Tor probes as such, not even when they are received by the HTTPS port, which removes the outer TLS layer. The first message that a Tor client sends after the TLS handshake is a `VERSIONS` cell, which happens to start with a '\0' byte, causing the payload to be truncated to a length of zero in the server log.

## 4.4 Counterprobing

Active probers seem to share their IP address pools with normal Internet users. To investigate this, we scanned some probers repeatedly using network diagnostic tools such as ping, traceroute, and Nmap. We started the first scan the moment a prober showed up. From then on, we repeated the scan hourly for 24 hours. Interestingly, the very first scan never yielded anything. The probers were unresponsive to all packets. Later scans, however, painted a different picture. In many cases, our port scans identified the IP addresses that were used for active probing just hours before as various versions of Windows and Linux. We found open ports used for file sharing, FTP, HTTP, RPC, and many other services typical for home users. This indicates that in many cases active probers share their address pools with Internet users. We were able to run a small number of counterprobes from a host within China, and the results matched those of simultaneous counterprobes from outside the country: prober IP addresses were initially unresponsive, but occasionally became responsive as disparate and seemingly ordinary Internet hosts.

With the previous experiments in hand, we then set out to develop tests to probe the architecture of the passive monitor that triggers the probes, and the active component responsible for sending them. A unidirectional three-packet sequence—SYN, ACK, and a Tor TLS client handshake packet—sent from a Chinese host suffices to trigger an active probe.

We built several tests and ran them from a Unicom host and a CERNET host to our EC2 infrastructure. These tests included a traceroute for the monitor; a fragmentation test that splits the request across multiple TCP packets to expose whether the passive detection maintains per-flow state; a SYN-ACK traceroute that uses TTL-limited packets to respond to probe requests; and a "milker" that repeatedly sends triggering requests to a Tor bridge.

| Client | Succeeded | Failed | Total |
|---|---|---|---|
| CERNET vanilla | 639 (12%) | 4,490 (88%) | 5,129 |
| Unicom vanilla | 90 (2%) | 3,864 (98%) | 3,954 |
| CERNET obfs2 | 4,584 (98%) | 81 (2%) | 4,665 |
| Unicom obfs2 | 4,153 (89%) | 515 (11%) | 4,668 |
| CERNET obfs3 | 5,015 (98%) | 95 (2%) | 5,110 |
| Unicom obfs3 | 3,402 (86%) | 552 (14%) | 3,954 |

Table 3: Connection success statistics for our clients in China. While vanilla Tor is mostly unreachable, obfuscation protocols are mostly reachable.

## 5. ANALYSIS

In this section, we study the following questions:

- Is active probing successful at discovering Tor bridges? (§ 5.1)
- What is the delay between a connection attempt to a Tor bridge and subsequent active probing of the bridge? (§ 5.2)
- Where in the network are the probes coming from? (§ 5.3)
- What types of probers do we see? (§ 5.4)
- Do active probers have "fingerprints" that distinguish them from normal clients? (§ 5.5)
- What is the underlying architecture of the probing system? (§ 5.6)

### 5.1 Effectiveness of Active Probing

To determine whether China's active probing infrastructure is effectively discovering and blocking Tor bridges, we analyzed the log files of our Tor clients inside China to see which connection attempts were successful in connecting to their assigned bridge. We consider a connection unblocked if the TCP handshake succeeded, because previous work showed that the Chinese censors block Tor bridges by dropping SYN-ACK segments [32]; if a bridge is blocked, it will be blocked at the TCP layer. We did this for each of the protocols supported by our clients and bridges: "vanilla" Tor, and the obfs2 and obfs3 obfuscation protocols.

Table 3 shows the results of this experiment. Our results show that obfs2 and obfs3 were almost always reachable for our clients both in CERNET and Unicom. This result is surprising because the Great Firewall has the ability to probe for obfs2 and obfs3 (Section 4.3). Vanilla Tor, on the other hand, is almost completely blocked. Figure 4 illustrates our attempts to connect to Tor bridges using vanilla Tor clients over time. We find that although Tor is mostly blocked, Tor clients succeed in connecting *roughly every 25 hours*. We believe that this reflects an implementation artifact of the GFW (e.g., many commercial firewalls "fail open" when their access control lists are being updated).

### 5.2 Delay Between Connection and Probing

We investigate how long it takes the Chinese censors to probe a Tor bridge after we initiate a connection to it. In 2011, Wilde observed that active probing occurred in 15-minute intervals [30], suggesting that the censors maintained a "probing queue" that was processed every 15 minutes. The

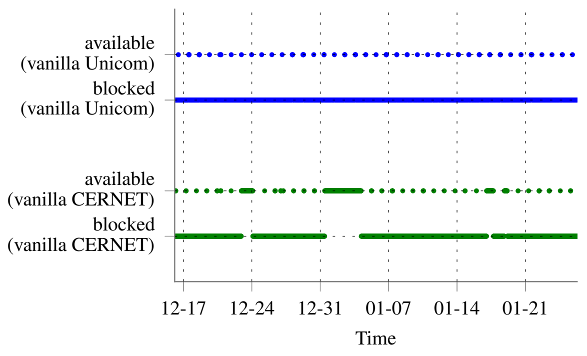


Figure 4: Successful Tor connections of our vanilla Tor clients in China over time. Every dot represents one connection attempt.

Sybil experiment helps us determine whether this is still the case. We calculated the time between when we established a connection to our Sybil node in France and when we observed the Chinese censors' subsequent probing connections. Figure 5 illustrates the results of this experiment. It shows the time difference in seconds (Y-axis) for every port (X-axis) on our decoy bridge. There are two interesting aspects. First, 56% of all probing connections arrived after less than one second (median 552 ms). We conclude that the Chinese censors have abandoned a 15-minute queue and now operate in real time. Second, there are several curious delay spikes where the Chinese censors took more than one minute to probe our bridge. We removed four outliers that had a probing delay above 800 seconds. Interestingly, all spikes decreased linearly until they reached the default of real-time probing.

Figure 6 shows a superset of the same data, but with the bridge port on the Y-axis and the absolute time on the X-axis. The line to the left consists of our decoy connections (red circles), which were quickly followed by probing connections (black crosses), which is basically the data shown in Figure 5. What Figure 5 did not show is the line to the right, a secondary "swarm" of probing connections. We captured these connections approximately 12 hours after our initial decoy connections (at which point our bridge was no longer running). The probers tried to connect several times, without success. Presumably, the Chinese censors were reconnecting to all bridges in order to verify whether they were still online.

### 5.3 Origin of the Probers

In total, we collected 16,083 unique prober IP addresses. There are 158 IP addresses in the Shadow dataset; 1,182 in the Sybil; and 14,912 in the Log. 111 of the addresses appear in two of the three datasets, and just one—202.108.181.70—appears in all three.

Prober IP addresses are rarely reused. Consider just the Log experiment: it recorded 16,464 probes carrying 15,249 distinct payloads; these probes were sent by 14,163 distinct IP addresses. 95% of addresses appear only once; another 4% appear twice. This lack of IP address reuse precludes simple blacklisting as a counter to active probing: a blacklisted prober IP is unlikely to be seen again. There is one clear outlier among probers, the aforementioned

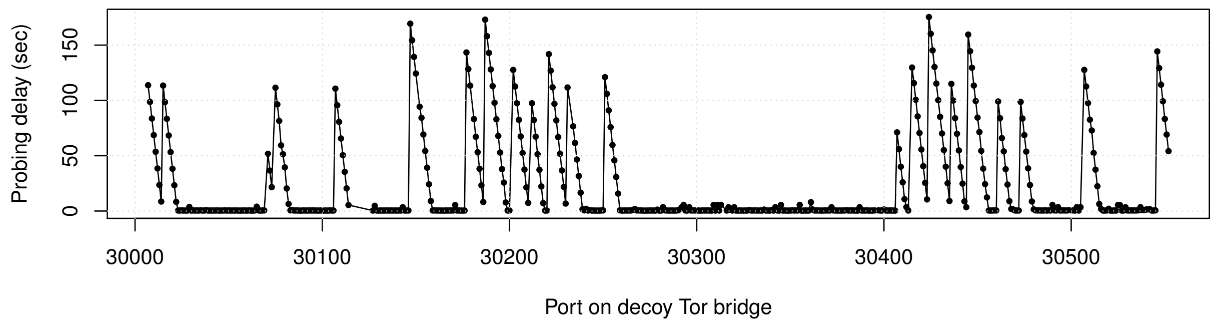


Figure 5: The time difference between decoy connections through our Sybil node and probing connections for every port on our Tor decoy bridge; the censors probe the ports on this bridge in rapidly ascending order. While the censors probed most ports immediately, some port ranges were only scanned significantly later.

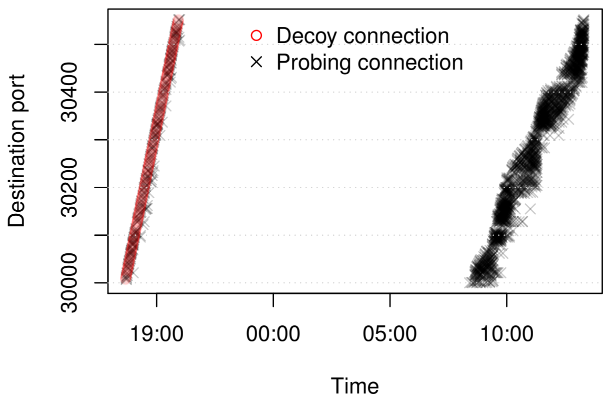


Figure 6: The arrival rate of probing connections. Decoy connections through our Sybil node are immediately followed by probing connections from the Chinese censors (the red circles and first set of black crosses are nearly on top of one another). After approximately 12 hours, the Chinese censors probed the same port range again.

| Count | SOA |
|---|---|
| 7,379 | ns1.apnic.net |
| 2,013 | *none* |
| 501–1,000 | ns.jlccptt.net.cn, ns.zjnbptt.net.cn |
| 201–500 | ns.sdjnptt.net.cn, ns3.tpt.net.cn, soa, ns.sxtyptt.net.cn, ns.hbwhptt.net.cn, ns.bta.net.cn, ns.hazzptt.net.cn, hzdns.zjnetcom.com, dns.fz.fj.cn, nmdns.hh.nm.cn |
| 101–200 | ns.timeson.com.cn, dnssvr1.169ol.com, HNGTDNS2.hunan.unicom.com, HNGTDNS1.hunan.unicom.com, nmc1.ptt.js.cn, NS2.NET.EDU.CN, ns.dcb.ln.cn, ns1.jscnc.net, ns.yn.cninfo.net, ns1.ah.cnuninet.net, localhost.localdomain |
| 1–100 | *53 others* |

Table 4: DNS Start of Authority of prober IP addresses.

202.108.181.70, which appears 248 times in the Log dataset. This special IP address has previously been associated with the GFW's active probing by Winter and Lindskog [32] and Majkowski [16]. We have observed it to send the obfs2, obfs3, and TLS probe types. It lies in AS4837, along with many other probers. No other prober IP address is in its /16 network.

Various measurements place the probing IP addresses almost entirely within China. Figure 7 shows the distribution of autonomous systems of prober IP addresses. Nearly every probe is from a Chinese AS. Table 4 shows the DNS Start of Authority (SOA) of the prober IP addresses. The most common SOA is ns1.apnic.net (Asia Pacific Network Information Centre, the regional Internet registry for the Asia Pacific region). The next most common SOAs are .cn domains and Chinese ISPs.

## 5.4 Probe Types

We categorized the probes we received into probe types. We discovered five distinct probe types across all four of our experiments, plus one "TLS" pseudo-type, as we describe below. Although the probes differ in important ways, they have in common the targeting of circumvention protocols. Section 5.5 describes how seemingly independent probes share low-level features (suggesting that they may originate from shared physical infrastructure), and highlights some of the features that may be useful for distinguishing active probers from genuine clients.

### *TLS.*

The "TLS" probe type is a TLS client hello, one whose application payload we did observe, or one that did not match any more specific probe type. This was the case, for example, in the Log experiment when a TLS probe arrived at a plaintext port: the application log records the beginning of the TLS header but nothing else. "TLS" probes could have been one of our other known types—Tor, SoftEther, or Appspot—or something else entirely.

### *Tor.*

The "Tor" probe type is a Tor `VERSIONS` cell received within a TLS connection. The probes we observed used a relatively obsolete "v2 handshake," superseded since 2011 [28]. This probe type is presumably aimed at the discovery of Tor bridges.

### *Obfs2.*

The "obfs2" probe type is the client part of the handshake of obfs2. Recall that obfs2's handshake is intended to look

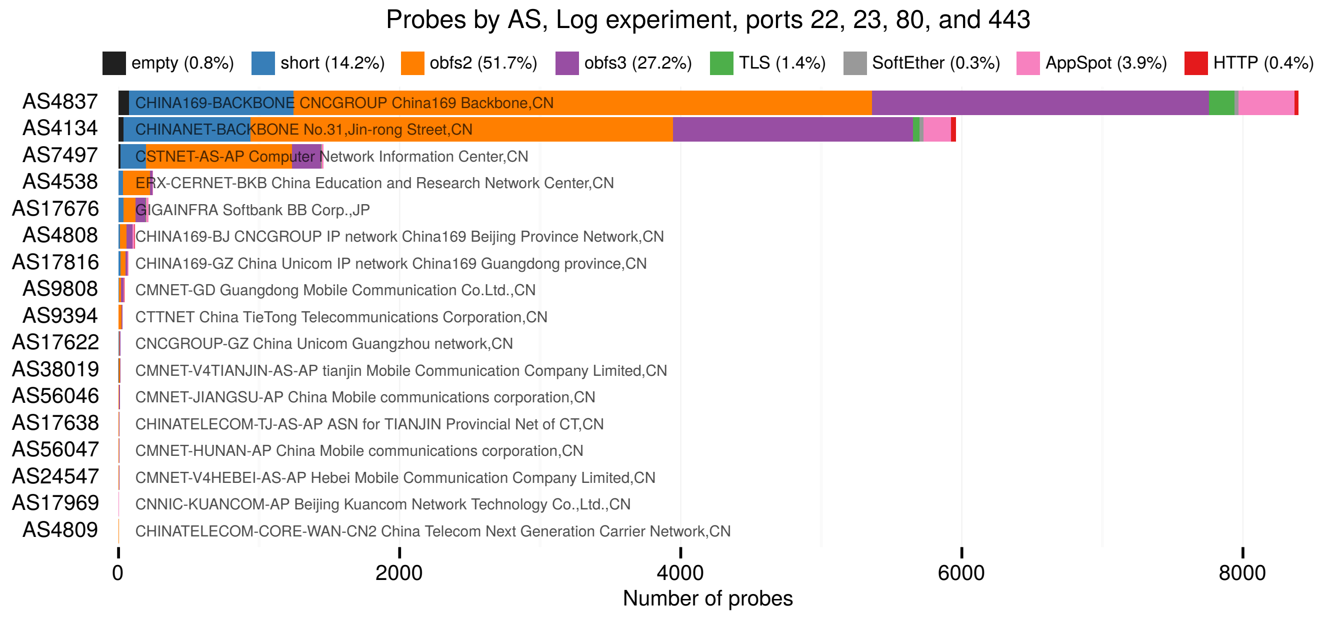


Figure 7: Three autonomous systems—AS4837, AS4134, and AS7497—account for 95% of probes observed in the Log experiment. We identified probing behavior by content, not by source address. Despite that, essentially all probes turn out to originate from China. All of the main ASes sent a variety of probe types; they do not appear to specialize. The most prolific prober, 202.108.181.70, lies in AS4837.

like a random bytestream; however its in-band key transmission makes it easy to identify, a great aid to our retrospective analysis. The first 20 bytes of a connection suffice to identify obfs2 with a negligible error rate.

### *Obfs3.*

The "obfs3" probe type is an obfs3 client handshake message. By design, obfs3 is resistant to passive detection, which poses problems for our retroactive analysis. Though we have many samples of probes whose properties are consistent with obfs3—that is, they appear random, of length between 192 and 8,386 bytes, and are not obfs2—it is not possible to say with certainty that they are obfs3 probes (and not, for example, some other random-looking protocol). To test our guess that the probes were probably obfs3, for a short period (Feb. 3–19, 2015) we enabled an obfs3 listener on the multi-protocol honeypot running on the server of the Log experiment. The listener would complete the server half of the obfs3 handshake, and then log everything received within the encrypted channel. By participating in the handshake, we found that the probes we would have suspected of being obfs3 were, in fact, obfs3. We have therefore labeled all probes bearing such as signature "obfs3," even though it is confirmed to be the case only in a small fraction of cases.

### *SoftEther.*

The "SoftEther" probe type is an HTTPS POST request: "`POST /vpnsvc/connect.cgi HTTP/1.1`". It resembles the first part of the client handshake of SoftEther VPN, the software that powers the VPN Gate circumvention system [21]. We discovered this probe type because it appeared many times in our HTTPS log; and we were able to observe it in detail (including its POST body) when it arrived on port 23 during the time we were running the multi-protocol honeypot listener. We further conjecture that some of the "TLS" probes that arrived at port 80 were in fact SoftEther probes, because they share a TCP timestamp sequence with other SoftEther probes. We first observed SoftEther probes in Aug. 2014. This is five months after the first active probes seen by the creators of VPN Gate, shortly after the release of their software [21].

### *AppSpot.*

The "AppSpot" probe type is an HTTPS GET request with a special `Host` header:

```
GET / HTTP/1.1
Host: webncsproxyXX.appspot.com
```

The '`XX`' is a number that varies across probes. The `Host` header reveals that this probe type is likely intended to discover servers that are capable of providing access to Google App Engine through domain fronting [8]. When a typical web server receives a request with an alien `Host` header such as this, it will respond with its default document, or an error message. But when a Google server receives the request, it will forward the request to the web application running at `webncsproxyXX.appspot.com`. Circumventors can and do run various proxies on App Engine using precisely this technique. These probes would be useful for eliminating any gaps left in the GFW's near-total block of Google services. We observed this probe type only on port 443. Though originating from the same pool of IP addresses as the other probe types, this one seems somewhat independent, in its TLS fingerprint, the rate of its TCP timestamp counter (Figure 11c), and the temporal patterns in its activity (Figure 8).

We crawled `webncsproxyXX.appspot.com` for values of `XX` between 1 and 100. Some of them were instances of GoAgent (a circumvention tool based on App Engine), while others are instances of a web-based proxy. This probe type exists in a few variations. For a time, they probes switched from requesting `/` to requesting `/twitter.com`, which would have caused the `webncsproxy` application to retrieve the Twitter home page. In July 2015, AppSpot probes began to arrive in pairs separated by a few seconds and originating from different IP addresses. The second probe in a pair had a different set of header fields; it also had a similar `Host: webncsproxyXX.appspot.com` header, though the value of '`XX`' was generally different from that of the first probe.

Figure 8 shows the complete probe history of the HTTP and HTTPS ports in the Log dataset. It is apparent that obfs2 and obfs3 are temporally related, and that "short" probes of less than 20 bytes also follow the same temporal pattern. The volume of TLS and SoftEther probes is much less than that of the other probe types. AppSpot appears to follow its own independent pattern. There are conspicuous lulls in probing behavior: between Dec. 2013 and Aug. 2014; between Feb. and Mar. 2015; and after May 2015. We do not know why probing nearly ceased during these periods.

## 5.5 Fingerprinting Active Probers

We now seek out telltale fingerprints in active probing: clues that may help distinguish probers from genuine clients, as well as clues as to how the probing infrastructure is implemented. We proceed layer by layer, starting with the IP layer and moving up through several application layers. Our analysis shows that despite there being a large number of probing IP addresses, there are likely only *a small number of independent processes* controlling them. In particular, our analysis of TCP initial sequence numbers and timestamps shows that shared state exists between disparate probing IP addresses.

### *IP Layer.*

In 2014, anonymous researchers inferred the structure of a DNS-poisoning censor node by analyzing side channels in the IP TTL and IP ID [1]. Figure 9 shows the TTL distribution of all SYN segments we received from probers in our Sybil experiment. Five TTL values account for the overwhelming majority of observed TTLs. We did not, however, find any discernible patterns in the distribution of the IP ID field.

### *TCP Layer.*

The TCP header is rich with potentially fingerprintable fields, particularly initial sequence numbers (ISNs), source ports, timestamps, and options. Patterns across TCP connections initiated by different IP addresses indicate that all the traffic originates from only a few processes (two in the Shadow dataset, one in the Sybil dataset, and about a dozen in the Log dataset). We analyzed in detail the TCP headers of SYN segments received in the Sybil experiment and found that they are all very similar.

*TCP options*: With respect to TCP options, all SYN segments:

- Employ an MSS of either 1400 (88% of probes), or 1460 (12%).

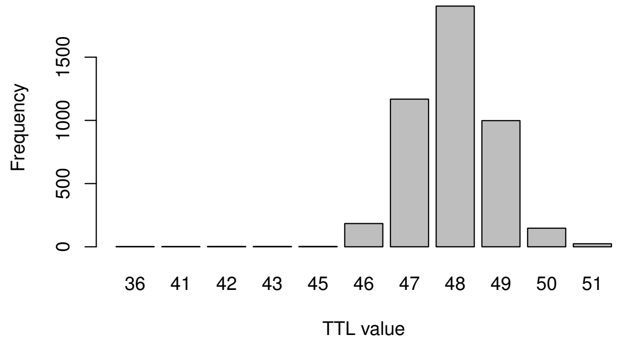


Figure 9: The TTL values in the SYN segments sent by probers. 99% of all observed TTL values are in between 45 and 51.

- Use window scaling of 7.
- Permit selective ACKs.
- Use the TCP timestamp option.
- Use the "no operation" option.

In particular, the OS identification tool p0f3 [35] yielded the following (truncated) signatures. The shaded part is identical in all three observed signatures.

`0:1460:mss*4,7:mss,sok,ts,nop,ws:df,id+:0` (1% of probes)
`0:1460:mss*20,7:mss,sok,ts,nop,ws:df,id+:0` (11% of probes)
`0:1400:mss*4,7:mss,sok,ts,nop,ws:df,id+:0` (88% of probes)

*Initial sequence numbers*: To protect TCP connections from off-path attackers, initial sequence numbers must not be guessable by an attacker. Modern operating systems use strong randomness to select ISNs. As a result, if all probing connections came from independent systems, we would expect no patterns in the distribution of ISNs.

We extracted the 32-bit ISNs of all SYN segments sent by probers. Figure 10 shows the ISN value on the Y-axis versus the time captured on the X-axis. To our surprise, the time series shows a striking, non-random pattern. Instead of uniformly distributed points over time, we see a "zigzag" pattern. ISNs increase until $2^{32}$ and then wrap around to 0. We conclude that the infrastructure derives ISNs from the current time.

The Sybil experiment induces a large amount of active probing over a short period of time—which was necessary for finding this ISN pattern. Our other experiments had too low a sample rate for the pattern to become apparent.

*Source ports*: We did not find any apparent patterns in the distribution of source ports. Interestingly, however, the source port distribution covers the entire 16-bit port range, including ports below 1024. This selection of ephemeral ports differs from that of standard operating systems, which typically only use port numbers above 1024.

*TCP timestamps*: We extracted the TSval from the TCP timestamp option [2] in all SYN segments sent by probers in the Shadow, Sybil, and Log experiments. Figure 11 illustrates the result. Though the SYN segments came from many different IP addresses, their timestamps form only a small number of distinct sequences (visible as straight lines in the graphs). We can characterize every line by its slope (i.e., its timestamp clock rate) and intercept (i.e., its system

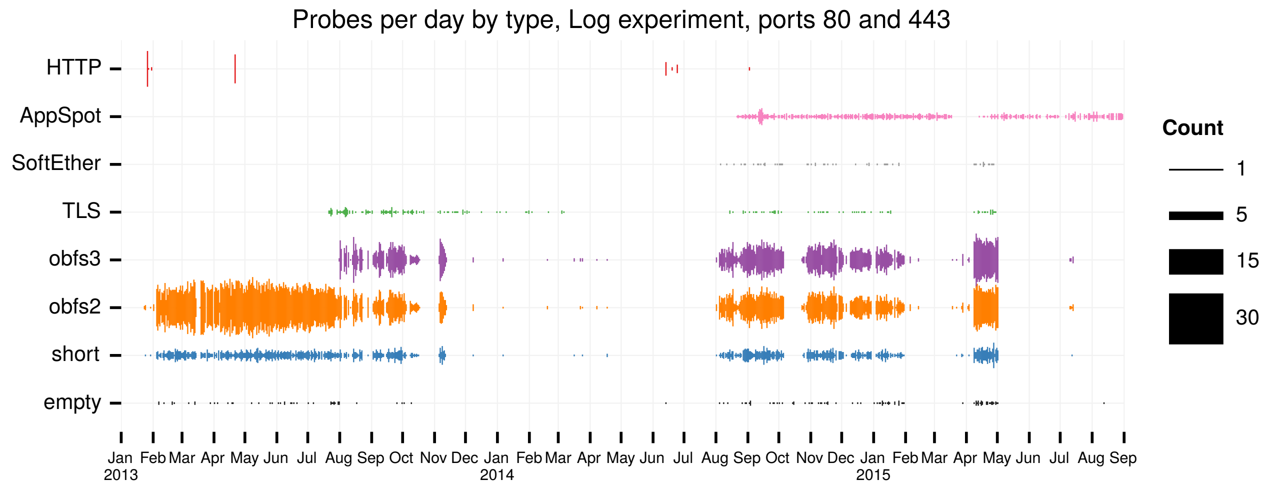


Figure 8: Probe types and volume of the HTTP and HTTPS ports in the Log experiment. The log file starts in Jan. 2011, but probes only began in 2013. The "HTTP," "AppSpot," and "SoftEther" rows are HTTPS requests to port 443; the others (including "TLS") are probes to port 80. The "short" probes are those that appear random, but are too short (< 20 bytes) for the obfs2 test. We believe that the "short" and "empty" probe types are actually truncated "obfs2" or "obfs3" probes. (Apache's log file truncates requests at the first '\0' or '\n' byte; a random byte string has about a 15% chance of being truncated in the first 20 bytes.) The "HTTP" row represents not probes, but ordinary requests for web pages on the server. They may be ordinary web users that happened to have an IP address that at another time sent some other type of probe; or they may be firewall operators web browsing from their probing infrastructure. (The requested pages were related to circumvention, which would be of interest to Chinese Internet users and firewall operators alike.) Two "HTTP" data points, at Aug. 2011, are not shown on the graph. They are from an IP address that would later send a "short" probe in May 2013.

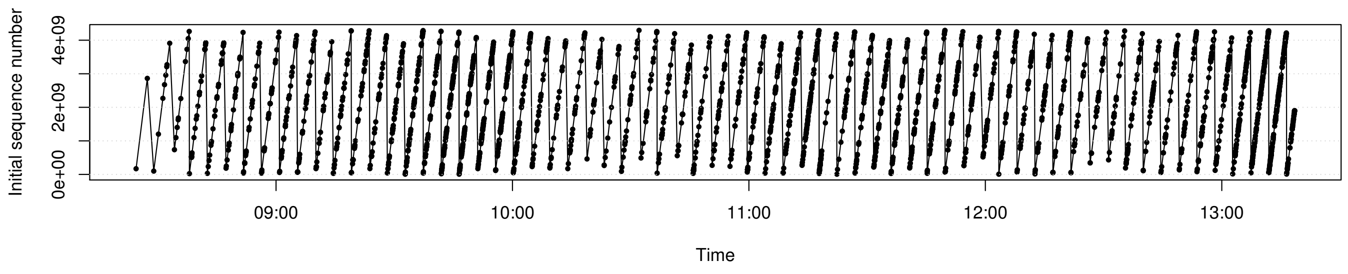


Figure 10: Initial sequence numbers of SYN segments sent by probers. Despite the probers coming from different IP addresses, a clear linear pattern manifests.

uptime). If all probes came from different physical systems, we would expect a larger number of distinct sequences.

### *SSL/TLS Layer.*

The Tor protocol is encapsulated within TLS. The TLS protocol has many features that enable fingerprinting. We analyze the TLS "client hello," the first message sent by a client—or an active prober—after establishing a TCP connection. We used a TLS fingerprinting patch [15] for the passive network fingerprinting tool p0f [35]. We captured a total of 621 client hellos in the Sybil dataset. They all had the same TLS fingerprint: TLSv1.0, with a particular list of 11 cipher suites, support for the TLS session ticket extension, and support for compression.

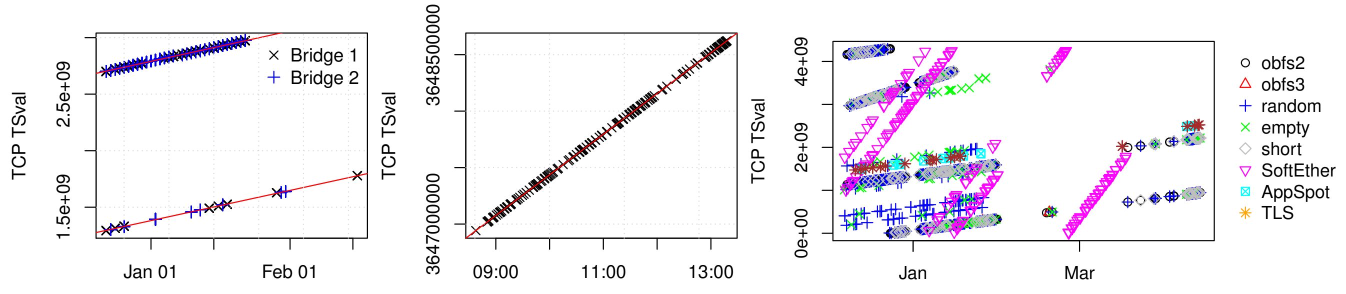
To better understand how common this fingerprint is, we extracted the offered cipher suites of all Tor clients connecting to a Tor guard relay under our control.[2] Over a 24-hour period, we observed 236,101 client hellos, out of which only 67 used the cipher suites listed above.

### *Application Layer—Tor.*

After the TLS handshake, a Tor client is supposed to send a `VERSIONS` cell [5, §4], in which it declares what versions of the Tor protocol it supports. After that there is further interaction before the establishment of a Tor circuit.

By inspecting the log files of a private Tor bridge, we found that probers send only the `VERSIONS` cell, and none of the other cells that would normally follow in order to establish a full Tor connection. After receiving a `VERSIONS` cell, our bridge, following the protocol specification, replied with a `NETINFO` cell, after which the probers abruptly closed the

[2]We extracted only the cipher suites and did not capture or store identifying information such as IP addresses.



(a) Shadow experiment. (b) Sybil experiment. (c) Log experiment.

Figure 11: The TCP TSval value of SYN segments received by active probers.

connection. The probers' `VERSIONS` cell declares support for Tor protocol versions 1 and 2, which were superseded in Oct. 2011 [18], and are now only supported for backward compatibility (the current protocol version is 4). The fact that probers use such an old protocol version, suggests, to us, that the Tor probes were developed in 2011 or earlier, and not materially updated since. We briefly modified a Tor bridge to ignore connections that offered only versions 1 and 2 and found that such a bridge does not get blocked despite being probed. (This is unfortunately not a universally deployable defense against probing because there are still some old clients that support only old protocol versions, and ignoring their handshakes would cut them out of the network.)

## *Application Layer—obfs2.*

The active probers' implementation of obfs2 conforms with the protocol specification: A 16-byte seed, an encrypted "magic" number, a padding length of 0–8,192 bytes, and random padding. The amount of random padding matches the declared padding length, except in a small number of cases when the TCP stream ended prematurely because of missing segments. The probers do not send anything in the encrypted payload layer inside the obfs2 obfuscation layer, even when communicating with an actual obfs2 server. In genuine use of Tor with obfs2, there is TLS within the obfs2 layer, but within the active probers' obfs2 there is no payload at all—the prober simply terminates the TCP connection. It seems that the mere existence of an obfs2 server is sufficient evidence for the censor.

Obfs2 clients should use fresh randomness for every connection. We were therefore surprised to find a few instances of duplicate obfs2 bytestreams. If two obfs2 streams have identical payloads, they must have had identical session keys and identical random padding. Out of 8,479 obfs2 probes received in the Log experiment, 56 (0.7%) were part of a pair having identical payloads. We did not find any payload that occurred more than twice. In every case, the paired probes came from different IP addresses, and arrived at nearly the same time, never more than five seconds apart. The probability that two independent obfs2 streams are identical is negligible; therefore the probers must share some state behind the scenes—at least a weak random number generator if not actually complete process state.

This apparent "state leakage" shows up in another way. Occasionally, an IP address that sent one half of a pair of identical obfs2 probes also sent, nearly simultaneously, a probe of some other type. Here is a sample from the Log experiment on port 80. Two different IP addresses sent the same obfs2 payload in the same second. One second later, one of the two additionally sent a TLS probe:

```
2014-08-29 15:44:01  60.216.143.31  obfs2  eef890766636...
2014-08-29 15:44:01  14.135.253.56  obfs2  eef890766636...
2014-08-29 15:44:02  14.135.253.56  tls    160301
```

Curiously, the active probers do more work to detect obfs2 than they strictly have to. Although the protocol does not require it, existing server implementations send their seed, magic number, and padding immediately upon receiving a TCP connection, without waiting the client's half of the handshake. A more stealthier active prober would open a TCP connection and simply listen. It would be able to detect current obfs2 servers without being so conspicuous.

## *Application Layer—obfs3.*

In contrast to obfs2, the active probers' implementation of obfs3 differs from the specification in an interesting way that does not affect protocol semantics. The specification calls for each side to send 0–8,194 bytes of random padding, but in two chunks, each with a length that is a uniform random number between 0 and 4,097. Instead, the active probers send their padding all at once, in a single chunk of length between 0 and 8,194. It is therefore possible to fingerprint active probers in 50% of cases: if the first padding chunk is longer than 4,097 bytes, then the client is an active prober.

As with obfs2, we observe some duplicated probe payloads. Out of 4,493 obfs3 probes received, 82 (1.8%) share their exact payload with another. The elements of a pair arrive within a few seconds of one another. As with obfs2, there is no payload within the obfuscation layer.

## *Application Layer—SoftEther.*

The SoftEther probe—an HTTPS POST request with a particular request body—matches one formerly sent by the genuine SoftEther VPN client. However, since July 2014, the genuine client has included a `Host` header containing the IP address of the VPN server. The active probers do not set this header, making them distinguishable.

There is another way to identify SoftEther probes. Although the SoftEther VPN protocol lacks documentation, in our experiments with version 4.15 we found that the client software always sends a GET request before sending its POST. The purpose of the GET request is to determine whether the server is SoftEther VPN and not some other HTTPS server. Because the active probers do not send the preceding GET request, we can distinguish them from legitimate clients.

The TLS fingerprint of the SoftEther probes differs strikingly from that of the actual SoftEther VPN client software, which has more and newer ciphersuites, and various extensions.

### *Application Layer—AppSpot.*

The special `Host` header of the AppSpot probe type is a dead giveaway to its purpose of discovering servers capable of fronting access to Google App Engine. All the probes we saw carry a fairly distinctive and specific `User-Agent` string, which is probably spoofed, as the rest of the header is inconsistent with its purported version of Chromium. The declared version of the browser was originally released in Apr. 2014, and superseded just two weeks later by a new update. We found a small number of real web requests using this `User-Agent`, but the great majority were active probers. The first AppSpot probes arrived in Sep. 2014.

Among other header inconsistencies, the probes set the header `Accept-Encoding: identity`, which forces the server to send the response body uncompressed. We used this characteristic to weed out the small number of non-prober requests that happened to use the same `User-Agent` string—these requests, using a real Chromium browser, would have set `Accept-Encoding: gzip`, and the server would have compressed its response. We can therefore identify active probes in our server logs because the number of transferred bytes is greater than it should be.

The TLS signature of AppSpot probes entirely differs from that of the claimed version of Chromium. The probes almost certainly reflect use of a custom program that merely imitates a web browser.

## 5.6 Characteristics of the Probing System

We designed our Counterprobe experiment (Section 4.4) to illuminate multiple features of both the active probing sensors and its probing network. We find clear evidence that the sensor responsible for triggering probes operates in a single-sided fashion, meaning that it only considers unidirectional flows. Our experiments showed that an unacknowledged series of a SYN segment, followed by an ACK, and finally data (i.e., Tor's TLS client hello) suffices to trigger a probe. The following subsections discuss additional findings.

### *The sensor does not process stateless segments.*

Some DPI sensors are stateless, i.e., they process TCP segments in isolation, without considering the TCP connection state. To learn if the active probing sensor is stateless, we set out to attract a probe in two ways: once after establishing a three-way handshake and once—on a different port—without prior handshake. The stateful data triggered a probe and the stateless did not. This matches our understanding of the behavior of the Great Firewall. However, it differs from the Great Cannon [17] that has been used to inject malicious JavaScript into web pages, which acts on naked packets.

### *The sensor does not seem to robustly reassemble TCP.*

Next, we tried to establish if the sensor is reassembling TCP streams. In the first step, we sent the triggering data in a single TCP segment after establishing a TCP connection, which, as expected, attracted an active probe. In the next step, we split the triggering data across packets in 20 byte increments—again after establishing a TCP connection. The fragmented data did not trigger an active probe, which differs from the GFW [13].

This behavior was already observed by Winter and Lindskog [32, §5.2] in 2012. There are, however, reports stating that the active probing sensor used to reassemble TCP streams at some point [31].

### *Traceroute to the sensors.*

We sent response-triggering packet trains with the TTL encoded in the port selection, and also performed a similar traceroute to locate the Great Firewall, from both a Unicom server and a CERNET server. Unicom's sensor appears to operate on the same link as the GFW, but the CERNET sensor appears one hop closer to our server.

Together, these three tests suggest that the active prober's sensor is distinct from both the RST-injecting portion of the Great Firewall and the sensor in the Great Cannon.

### *Inferring the physical infrastructure.*

Section 5.5 suggests that there is clearly a substantial amount of centralization, as probes from a diverse range of IP addresses share both TCP timestamps and initial sequence number patterns. But what is the nature of the IP addresses from which the probes originate? We envision three possibilities:

1. A network of distributed proxies that simply forwards raw packets, and is centrally controlled by the active probing system.

2. A few centralized packet injection devices that extract the probed server's reply via passive monitoring.

3. A few centralized man-in-the-middle devices that selectively intercept traffic, temporarily hijacking end-system IP addresses, in a manner similar to the Great Cannon.

Our solution to distinguish these three possibilities was to deploy a system that responds to incoming probes with a series of TTL-limited packets, effectively acting as a traceroute. Our responses included:

- SYN-ACK packets, encoding the hop in the sequence number.

- UDP packets, encoding the hop in the IP ID field.

- UDP packets to the probe's source.

- SYN-ACK (with the hop encoded in both the port and sequence number) and UDP packets to the topologically next IP address.

- SYN-ACK and UDP packets to the topologically next subnet.

We triggered probes by sending requests from our server in China, and our responses were sent blindly, only capturing packet traces for a post-processing analysis.

The resulting traceroutes argue against the possibility of packet injection: a packet injecting system is unable to suppress the legitimate reply, and we would expect to see ICMP time exceeded packets corresponding to the answered ACKs. The only exception would be if the injector's author maintained a careful topology, ensuring that the injector never replied to an observed packet with too low a TTL to reach the real destination. Given that other Chinese systems, including the detectors in the Great Cannon and the Great Firewall's RST injector, do not perform such an analysis, we find it unlikely to believe that this system used packet injection.

For the same reason the traceroutes argue against a Great Cannon-type interceptor: the UDP and TCP traceroutes are consistent for both for the target IP address and the next IP address in sequence. In particular, note that the SYN-ACK is never answered early. To be consistent with the next IP address's topology, again the probing devices would need a deep understanding of the actual network.

For both cases, such a deep understanding of the network's topology would not significantly increase the system's stealth: It's already clear that the probes come from thousands of different IP addresses, and probing itself is not, by its nature, stealthy. Thus we believe, but cannot prove conclusively, that the system conducts its probes through a large, distributed proxy network.

## 6. CONCLUSION

Our work paints a detailed picture of active probing, the Great Firewall's newest weapon in the arms race of Internet censorship. Our results show that the system operates in real time, but regularly suspends for a short amount of time. It is capable of detecting the servers of at least five circumvention protocols and is upgraded regularly. We show that the system makes use of a vast amount of IP addresses, provide evidence that all these IP addresses are centrally controlled, and determine the location of the Great Firewall's sensors.

Future work could develop more circumvention strategies that can defeat active probing. Fortunately, users behind the GFW already have a number of working circumvention tools at their disposal, and other designs are in development. A family of techniques known variously as decoy routing [12], end-to-middle proxying [34, 9], and domain fronting [8] colocate the circumvention system's entry point with important network infrastructure, so that it cannot be easily blocked even if its address is known.

The obfuscation protocols ScrambleSuit [33] and its successor obfs4 [27] tread a different path by requiring clients to prove knowledge of a shared secret before responding. This technique is essentially port knocking at the application layer. Other proposals would add scanning resistance at the TCP layer [23] or the Tor protocol layer [11].

Our datasets, code, and auxiliary information are available online at `https://nymity.ch/active-probing/`.

## Acknowledgments

This material is based upon work supported in part by the National Science Foundation under grant nos. #1223717, #1518918, #1540066, and #1518882. This work was also supported in part by funding from the Open Technology Fund through the Freedom2Connect Foundation and from the US Department of State, Bureau of Democracy, Human Rights and Labor. The opinions in this paper are those of the authors and do not necessarily reflect those of any funding agency or governmental organization.

## 7. REFERENCES

[1] Anonymous. Towards a comprehensive picture of the Great Firewall's DNS censorship. In *FOCI*. USENIX, 2014.

[2] David Borman, Bob Braden, Van Jacobson, and Richard Scheffenegger. TCP extensions for high performance. RFC 7323 (Proposed Standard), September 2014.

[3] Richard Clayton, Steven J. Murdoch, and Robert N. M. Watson. Ignoring the Great Firewall of China. In *PET*. Springer, 2006.

[4] Roger Dingledine. Obfsproxy: the next step in the censorship arms race. https://blog.torproject.org/blog/obfsproxy-next-step-censorship-arms-race, February 2012.

[5] Roger Dingledine and Nick Mathewson. Tor protocol specification. https://spec.torproject.org/tor-spec.

[6] Roger Dingledine and Nick Mathewson. Design of a blocking-resistant anonymity system. Technical report, The Tor Project, 2006.

[7] Roya Ensafi, Philipp Winter, Abdullah Mueen, and Jedidiah R. Crandall. Analyzing the Great Firewall of China over space and time. In *PETS*. De Gruyter Open, 2015.

[8] David Fifield, Chang Lan, Rod Hynes, Percy Wegmann, and Vern Paxson. Blocking-resistant communication through domain fronting. In *PETS*. De Gruyter Open, 2015.

[9] Amir Houmansadr, Giang T. K. Nguyen, Matthew Caesar, and Nikita Borisov. Cirripede: Circumvention infrastructure using router redirection with plausible deniability. In *CCS*, pages 187–200. ACM, 2011.

[10] Andrew Jacobs. China further tightens grip on the Internet. http://www.nytimes.com/2015/01/30/world/asia/china-clamps-down-still-harder-on-internet-access.html, 2015.

[11] George Kadianakis. Bridge client authorization based on a shared secret. https://gitweb.torproject.org/torspec.git/tree/proposals/190-shared-secret-bridge-authorization.txt, 2011.

[12] Josh Karlin, Daniel Ellard, Alden W. Jackson, Christine E. Jones, Greg Lauer, David P. Mankins, and W. Timothy Strayer. Decoy routing: Toward unblockable Internet communication. In *FOCI*. USENIX, 2011.

[13] Sheharbano Khattak, Mobin Javed, Philip D. Anderson, and Vern Paxson. Towards illuminating a censorship monitor's model to facilitate evasion. In *FOCI*. USENIX, 2013.

[14] Zhen Ling, Xinwen Fu, Wei Yu, Junzhou Luo, and Ming Yang. Extensive analysis and large-scale empirical evaluation of Tor bridge discovery. In *INFOCOM*. IEEE, 2012.

[15] Marek Majkowski. SSL fingerprinting for p0f. https://idea.popcount.org/2012-06-17-ssl-fingerprinting-for-p0f/, June 2012.

[16] Marek Majkowski. Fun with the Great Firewall. https://idea.popcount.org/2013-07-11-fun-with-the-great-firewall/, July 2013.

[17] Bill Marczak, Nicholas Weaver, Jakub Dalek, Roya Ensafi, David Fifield, Sarah McKune, Arn Rey, John Scott-Railton, and Ron Deibert. An analysis of China's "Great Cannon". In *FOCI*. USENIX, 2015.

[18] Nick Mathewson. Proposed version-3 link handshake for Tor. https://gitweb.torproject.org/torspec.git/tree/proposals/176-revising-handshake.txt, January 2011.

[19] Jon McLachlan and Nicholas Hopper. On the risks of serving whenever you surf: Vulnerabilities in Tor's blocking resistance design. In *WPES*. ACM, 2009.

[20] Leif Nixon. Some observations on the Great Firewall of China. https://www.nsc.liu.se/~nixon/sshprobes.html, 2011.

[21] Daiyuu Nobori and Yasushi Shinjo. VPN gate: A volunteer-organized public VPN relay system with blocking resistance for bypassing government censorship firewalls. In *NSDI*. USENIX, 2014.

[22] Jong Chun Park and Jedidiah R. Crandall. Empirical study of a national-scale distributed intrusion detection system: Backbone-level filtering of HTML responses in China. In *ICDCS*. IEEE, 2010.

[23] Rob Smits, Divam Jain, Sarah Pidcock, Ian Goldberg, and Urs Hengartner. BridgeSPA: Improving Tor bridges with single packet authorization. In *WPES*. ACM, 2011.

[24] Sparks, Neo, Tank, Smith, and Dozer. The collateral damage of Internet censorship by DNS injection. *SIGCOMM Computer Communication Review*, 42(3):21–27, 2012.

[25] The Tor Project. obfs2 (the twobfuscator). https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs2/obfs2-protocol-spec.txt.

[26] The Tor Project. obfs3 (the threebfuscator). https://gitweb.torproject.org/pluggable-transports/obfsproxy.git/tree/doc/obfs3/obfs3-protocol-spec.txt.

[27] The Tor Project. obfs4 (the obfourscator). https://gitweb.torproject.org/pluggable-transports/obfs4.git/tree/doc/obfs4-spec.txt.

[28] The Tor Project. TLSHistory. https://trac.torproject.org/projects/tor/wiki/org/projects/Tor/TLSHistory.

[29] The Tor Project. Tor: Pluggable transports. https://www.torproject.org/docs/pluggable-transports.html.en.

[30] Tim Wilde. Knock knock knockin' on bridges' doors. https://blog.torproject.org/blog/knock-knock-knockin-bridges-doors, 2012.

[31] Philipp Winter. #8591: GFW actively probes obfs2 bridges. https://bugs.torproject.org/8591, March 2013.

[32] Philipp Winter and Stefan Lindskog. How the Great Firewall of China is blocking Tor. In *FOCI*. USENIX, 2012.

[33] Philipp Winter, Tobias Pulls, and Juergen Fuss. ScrambleSuit: A polymorphic network protocol to circumvent censorship. In *WPES*. ACM, 2013.

[34] Eric Wustrow, Scott Wolchok, Ian Goldberg, and J. Alex Halderman. Telex: Anticensorship in the network infrastructure. In *USENIX Security Symposium*. USENIX, 2011.

[35] Michal Zalewski. p0f v3 (version 3.08b). http://lcamtuf.coredump.cx/p0f3/, 2014.